Utveckling av ett webbaserat frakthanteringssystem

Robin Larsson

Examensarbete för ingenjör (YH)-examen

Utbildningsprogrammet för informationsteknik

Vasa 2016

EXAMENSARBETE

Författare: Robin Larsson

Utbildningsprogram och ort: Informationsteknik, Vasa

Handledare: Kaj Wikman

Titel: *Utveckling av ett webbaserat frakthanteringssystem*

Datum: 10.5.2016 Sidantal: 32

Abstrakt

Detta examensarbete handlar om att skapa en mobilvänlig webbapplikation för att ersätta ett existerande pappersbaserat system, som används av en speditionsfirma. Webbapplikationen använder sig av webbtekniker såsom HTML och JavaScript, samt tekniker som Web API 2, Entity Framework och SQL för att hantera databasdelen. Som värd för webbapplikationen kommer Windows Azure att användas, vilket medför en hög tillgänglighet av webbapplikationen.

Resultatet blev en fungerande webbapplikation som helt ersatt det tidigare systemet och samtidigt erbjuder några nya hjälpmedel för att underlätta det dagliga arbetet för speditören, såsom ett dagsschema som kan editeras.

Språk: svenska Nyckelord: ASP.net, Web API 2, Knockout, Durandal

---

OPINNÄYTETYÖ

Tekijä: Robin Larsson

Koulutusohjelma ja paikkakunta: Informaatiotekniikka, Vaasa

Ohjaaja: Kaj Wikman

Nimike: *Web-käyttöisen rahdinkäsittelyjärjestelmän kehitys*

Päivämäärä: 10.5.2016 Sivumäärä: 32

Tiivistelmä

Tämä opinnäytetyö käsittelee mobiiliystävällisen web-sovelluksen kehittämistä. Sovellus korvaa nykyisen paperikäyttöisen järjestelmän, joka on käytössä kuljetusfirmalla. Web-sovellus käyttää web-tekniikoita, kuten HTML:ää ja JavaScriptiä, sekä myös tekniikoita kuten Web Api 2:a, Entity Frameworkia ja SQL:ää hoitaakseen tietokantapuolen. Itse web-sovellus tule olemaan sijoitettu Windows Azureen, mikä mahdollistaa korkean luotettavuuden ja saatavuuden.

Tuloksena oli tarkoitukseen sopiva ja toimiva web-sovellus, joka on kokonaan korvannut edellisen järjestelmän, ja samalla tarjoaa lisäapuja, jotka tekevät päivittäisestä työstä helpompaa kuljetusfirmalle. Esimerkkinä tästä on editoitava päiväjärjestelmä.

Kieli: ruotsi Avainsanat: ASP.net, Web API 2, Knockout, Durandal

---

BACHELOR'S THESIS

Author:	Robin Larsson
Degree Programme:	Information Technology, Vaasa
Supervisors:	Kaj Wikman

Title*: Development of a Web-based Freight Management System*

Date: 10.5.2016 Number of pages: 32

Abstract

This thesis is about creating a mobile friendly web application, intended to replace the existing paper based system that is in use by a forwarding agency. The Web application utilizes web techniques, such as html and JavaScript, as well as Web Api 2, Entity Framework and SQL for database usage. The web application will be hosted in the cloud, using Windows Azure, which provides high availability.

The result of this thesis is a functioning web application that has completely replaced the existing system, and provides some useful additional features to help the everyday work for the user. One example of this is the editable day schedule.

Language: Swedish Key words: ASP.net, Web API 2, Knockout, Durandal

---

Innehållsförteckning

Förklaringar

DOM	Document Object Model. En träd-strukturell representation av alla element på en webbsida.
AMD	Asyncronous Module Definition
.NET Ramverk	En samling komponenter som finns i Windows operativsystemen, som klarar av att exekvera kod som är skriven för respektive komponent.
Integer-värde	Integer är en datatyp som används inom programmering, och representerar ett numeriskt värde.
Array	En sorts samling eller uppradning av variabler, som används inom programmering.
SPA	Single Page Application.
API	Application programming interface.

1 Inledning

Detta inledande kapitel kommer att presentera uppdragsgivaren, samt ge en inblick över bakgrunden till detta examensarbete.

1.1 Uppdragsgivare

Ibiworks Ab är ett företag i Jakobstad som i nuläget har tre ordinarie anställda. Verksamheten består till stor del av konsultuppgifter, såsom införskaffning och upprätthållande av olika programtjänster åt andra företag. Ett exempel på en sådan tjänst är Office365 som Ibiworks har tagit i bruk och underhåller hos ett flertal kunder. Office365 innehåller förutom de populära kalkyl- och textredigeringsprogrammen också e-posttjänster för företag, samt olika fildelnings- och samarbetstjänster.

Andra konsultuppgifter som man erbjuder är t.ex. anskaffning av servrar åt företag, samt konfigurering av dessa enligt önskemål.

Ibiworks utför även programutveckling, främst med hjälp av tekniker som finns i Microsofts .NET ramverk men också webb- och skriptbaserat. Självutvecklade program medför även vidare underhåll, och utveckling.

1.2 Bakgrund

Grundidén till examensarbetet uppstod efter att en speditionsfirma närmat sig uppdragsgivaren och visat hur dess orderhanteringssystem såg ut. Det så kallade systemet som speditionsfirman använde, var att manuellt med papper och penna skriva ner och planera sitt dagsschema. Under dagens lopp fick man in flera beställningar på leveranser och dessa antecknades vartefter. Chaufförен var också tvungen att fylla i fraktsedlar för varje avsändare och mottagare manuellt, vilket också betydde extra tid och arbete.

Uppdragsgivaren kunde konstatera att en applikation för ändamålet vore en lämplig lösning.

2 Uppgift

Detta kapitel kommer att beskriva hur den tidigare lösningen såg ut genom att beskriva varje steg i den, samt beskriva syftet med uppdraget.

2.1 Tidigare lösning

Den tidigare lösningen bestod som sagt av ett manuellt papperssystem, med ett par olika skeden. Detta underkapitel presenterar dessa skeden.

2.1.1 Anteckning av beställningar

Allt började från att speditören fick ett samtal, med en beställning av en leverans för en kund. Dessa leveranser antecknades ner på ett papper med en avsändare, mottagare, och en betalare. En beställning hade också andra uppgifter, såsom ett upphämtningsdatum och eventuellt en fraktsedelnummer, ifall beställaren använde en egen fraktsedel.

2.1.2 Planering av rutt

Speditören hade vid början av dagen förhoppningsvis redan ett par leveranser att hämta, som blivit beställda föregående dag. Dessa planerades i förväg in på en lista, där upphämtningen av leveranserna ordnades enligt vad som ansågs vara den mest lämpliga rutten. I praktiken var dessa endast grupperade enligt upphämtningsstad, och tidpunkt. Till exempel Karleby 09:00. Under denna grupp skrevs kundernas namn in, och gruppen kunde i bästa fall innehålla ett tiotal beställningar, vars upphämtningsordning endast fanns i huvudet på speditören. Detta medförde att beställningar ibland blev försenade, eller i värsta fall glömdes bort.

2.1.3 Ifyllning av fraktsedlar

Vid upphämtning och leverans till en kund krävdes i de flesta fall att en fraktsedel skulle skrivas ut. Undantag till detta var ifall kunden hade ofta återkommande leveranser, och i så fall hade kommit överens om att leveranserna inte krävde fraktsedel. I detta fall användes endast en för kunden specifik lista. Om fraktsedel krävdes blev det totalt tre papperslappar som skulle fyllas i.

2.1.4 Bifogning av fraktsedlar vid fakturering

De flesta av speditörens kunder ville bli fakturerade i slutet av månaden. Detta betydde att ett flertal leveranser skulle ingå i denna faktura. För att kunna fakturera krävs att fakturan har en faktureringsgrund. Detta betydde i detta fall att speditören vid faktureringen var tvungen att summera upp alla leveranser för respektive kund, samt bifoga fraktsedlar för var och en av dessa leveranser.

För kunder med vilka man hade kommit överens om att inga fraktsedlar behövde skrivas ut, räckte det med att bifoga listan över alla dessa leveranser.

2.2 Syfte

Syftet med uppdraget var att utveckla en mobil webbapplikation, som skulle kunna användas från en mobil enhet i form av en tablett, eller mobiltelefon, och som skulle ersätta det tidigare använda skriftliga orderhanteringssystemet. Mobiliteten var ett krav på grund av att speditören ville kunna använda applikationen från sin bil under tiden som leveranser gjordes.

Genom att göra en sådan applikation skulle speditören kunna spara tid och arbete vid inmatning av nya beställningar, vid fakturering, samt lättare kunna planera sitt körschema för dagen.

Uppdragsgivaren hade också som syfte att med hjälp av projektet, få en bättre inblick i hur utvecklingsplattformen, det vill säga en webbapplikation tillgänglig i molnet, och använder sig utav webbtekniker, lämpar sig för andra liknande applikationer. En ytterligare möjlighet var dessutom att ganska enkelt kunna modifiera applikationen för att passa in ett annat liknande scenario.

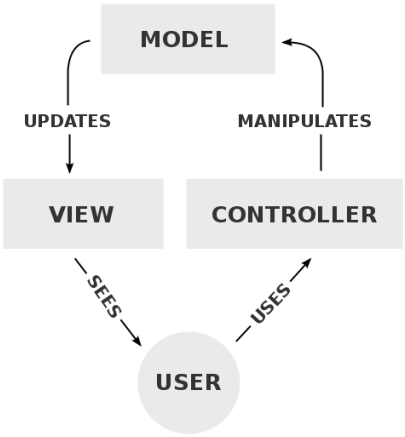
3 Använda tekniker

Detta kapitel kommer att gå igenom och förklara de i applikationen använda teknikerna. Inledningsvis kan nämnas, att webbapplikationen utvecklades med Visual Studio 2013, och använder sig utav ASP.NET MVC 5, och Web API 2-ramverken för att hantera information mellan användaren och databasen. Som kodningsspråk för dessa används C Sharp. Användargränssnittet är byggt med hjälp av Durandaljs, som är ett JavaScript-ramverk för att underlätta att bygga Single Page Applikationer.

3.1 ASP.NET MVC 5

ASP.NET MVC är ett ramverk som används för att bygga webbapplikationer, där man önskar använda sig av en kodningsstruktur i form av Model-View-Controller. MVC separerar användargränssnittet i tre olika delar (se figur 1):

- **Model**: Innehåller klasser som representerar data som hanteras, samt regler för hur data tillåts att modifieras. Uppdaterar view när modellen själv ändrar.
- **View**: Beskriver hur data ska presenteras åt användaren. Uppbyggt som ett template, som uppdateras enligt modellen.
- **Controller**: Innehåller logik som hanterar användarens kommunikation, styr applikationens flöde, och annan diverse logik. Till exempel hämtar controllern data från databasen enligt vad användaren önskar, och uppdaterar modellen med denna data.

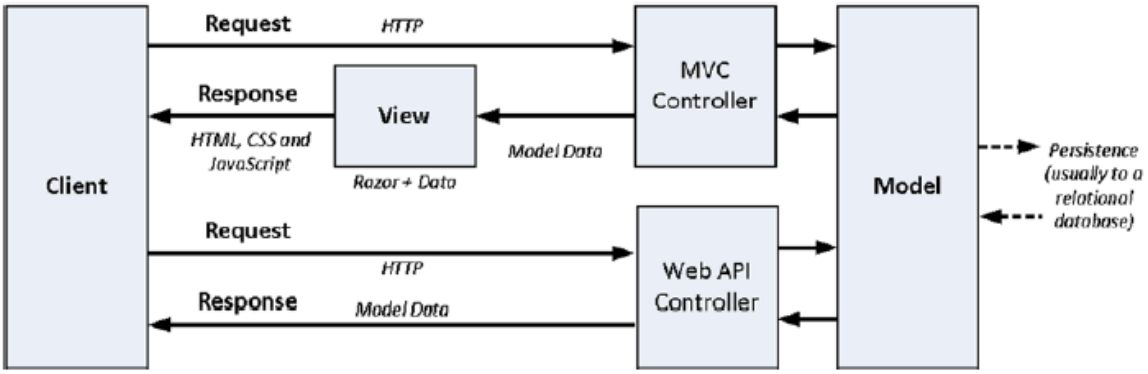


Figur 1. MVC-exempel.[18]

En webbapplikation har traditionellt ett flertal olika sidor. Dessa sidor har alla var sin View, tillhörande Model, och Controller.[17].

3.2 Web API 2

ASP.NET Web API 2, är likt MVC också ett ramverk som används för att bygga gränssnitt mot vilka man kan göra anrop, med till exempel JavaScript. Som svar på anropen returnerar Web API data, vars utformning beror på dess modell (se figur 2).



Figur 2. Illustrering över Web Api:s funktion.[16].

Till skillnad från MVC som också innehåller Controllers, returnerar en Web Api controller endast data, till skillnad från en MVC-controller som returnerar en vy som innehåller data.

Detta lämpar sig bättre när man vill skapa en Single Page Application (se kapitel 3.7), och själv genererar vyn, eller när man önskar returnera en fil som generas före den skickas ut.

3.3 HTML

HTML står för Hypertext Markup Language, och är standardmärkspråket som används för att skapa webbsidor. HTML är alltså inget programmeringsspråk, vilket betyder att man på ett mera betydelsefullt sätt kan uttrycka en webbsidas utformning, i motsats till om man använde ett programmeringsspråk. [10].

Ett HTML-dokument innehåller element, vilka skrivs med hjälp av taggar, till exempel ”<html></html>”. Element finns för olika typer av innehåll, till exempel knappar, tabeller, textfält osv. Ett av de vanligaste elementen är ” ”, som är en indelning av sidan och ofta innehåller underelement. ”<script>” element används för att bädda in JavaScript kod på en sida.

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

Figur 3. Exempel på HTML.

3.4 CSS

Cascading Style Sheets är ett formatmallsspråk, som används för att anpassa utseendet för ett dokument som skrivits med ett märkspråk. Oftast är detta märkspråk HTML, men CSS kan också tillämpas på andra typer av dokument, såsom till exempel XML.

Den största nyttan med CSS är att man får en separering mellan innehållet av ett dokument, och dokumentets design, som består av detaljer såsom layout, färger och fonter. Detta gör att man kan använda samma design över flera dokument, och mera flexibelt kan specificera en gemensam stil över flera likadana element i exempelvis ett HTML-dokument. [11].

CSS koden kan specificeras i en skild fil, eller bäddas in i ett HTML-dokument. Koden för CSS innehåller regler, som kan gälla ett visst eller flera element, eller ett element med en viss Id. En regel i CSS kan fastställa att till exempel alla h1-elements font typ ska vara fet, och ha en ram (se figur 4).

Figur 4. CSS-exempel.[11]

3.5 JavaScript

JavaScript är ett lättkört, objektorienterat programmeringsspråk, som är mest känd för dess användning på webbsidor, men som också används i andra sammanhang, såsom webbläsartillägg och spelprogrammering [12]. JavaScript har trots likheter i namn, och dess syntax, ingen relation till Java. JavaScript körs på klienten, det vill säga i webbläsaren och används för att programmera webbsidans beteende och utseende vid inträffandet av en händelse. Detta görs genom att JavaScript-koden antingen bäddas in i ett script-element i HTML-filen, eller att en JavaScript-fil läses in från HTML-filen. Oftast används det senare alternativet, på grund av att man vill kunna skilja på JavaScript-koden. Också färdiggjorda JavaScript-bibliotek, såsom till exempel jQuery, består av en eller flera JavaScriptfiler.

```
var ex;
ex = 10;
ex = "Kan också ha ett strängvärde!";

var ex2 = {
    objEgenskap1 : 1,
    objEgenskap2: 2,
    objEgenskap3: function () {
        return "En egenskap kan också returnera en funktion!";
    }
}

var ex3;

ex3 = ex2.objEgenskap1 + ex2["objEgenskap2"];
skrivut(ex3);
skrivut(ex2.objEgenskap3());

function skrivut(indata) {
    alert(indata);
}
```

Kodexempel 1. JavaScript-exempel.

Till skillnad från andra programmeringsspråk, såsom C och Java, är JavaScript ett dynamiskt programmeringsspråk. Detta betyder att JavaScript-kod inte kompileras före användning, utan funktioner som annars körs vid kompileringen, körs under användningen. Syntaxen påminner dock om C, vilket grundarna hade i åtanke för att locka C-programmerare att använda JavaScript. Andra skillnader som JavaScript har är till exempel att datatyper associeras med värden, istället för med variabler. Som exempel så kan samma variabel innehålla antingen ett strängvärde, eller ett numeriskt-värde (se kodexempel 1).

3.6 JSON

JSON står för JavaScript Object Notation, och är ett lätt dataväxlingsformat. JSON har utvecklats för att vara lätt att läsa för människor, samt lätt att läsa in, och generera för maskiner. Baserar sig som namnet antyder, på en JavaScript-standard. Från denna standard har man tagit strukturerna för att bygga upp objekt, samt matriser, och använt dessa för att bygga upp textsträngar. [13].

Objektet i detta sammanhang består av ett eller flera namn/värde – par. Dessa namn/värde – par omges av klamrarna ”{” och ”}”. Varje par separeras med ett komma.

JSON innehåller också matriser, dessa omges av ”[” och ”]”. Antingen kan grundobjektet innehålla en matris, eller så kan ett värdefält vara en matris, med flera underobjekt (se kodexempel 2).

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Kodexempel 2. JSON-exempel.

3.7 Single Page Application

En Single Page Applikation, förkortat SPA, och är en webbapplikation eller webbsida, där innehållet, som namnet antyder, passar in på en enda sida. Orsaken till att man enbart vill använda en sida, är att man vill uppnå en mera responsiv användarupplevelse, där man inte navigerar mellan flera olika sidor, som var och en laddas in, vilket medför långsamhet.

Hur detta åstadkoms i praktiken, är att all nödvändig kod, i form av HTML, JavaScript, och CSS, hämtas vid första laddningen av sidan. Efter detta manipuleras dynamiskt det som visas enligt vilken sida, eller vy som man navigerar till, eller enligt användarens begäran. Några uppdateringar av webbsidan förekommer inte. Om man navigerar mellan olika sidor utförs ofta någon sorts animation för att imitera ett sidbyte.

SPA-konceptet använder en liknande filosofi som MVC, när det gäller strukturingen av användargränssnittet. Det vill säga, det finns en **Modell**, en **Vy**, och en **Kontroller**. Skillnaden till MVC är att modellen och vyn finns, och hanteras på klienten. Detta medför en något mindre resurskrävande applikation med tanke på serversidan, eftersom en stor del av logiken flyttas till klienten.

För att utveckla en SPA, finns ett antal JavaScript ramverk som körs på klientsidan, det vill säga i webbläsaren, som kan användas. Ett exempel på ett sådant är Durandal. För att kunna beskriva Durandal krävs att några andra JavaScript-ramverk först tas upp.

3.7.1 jQuery

jQuery är ett JavaScript-bibliotek, som kan användas på de flesta webbplattformar, och används för att förenkla kodning med JavaScript i webbsidor. Några exempel där jQuery underlättar JavaScript-kodning;

- Val av element på en webbsida (se kodexempel 3)
- Hantering av händelser (se kodexempel 3)
- Utförande av Ajax-förfrågningar (se kodexempel 3)

```
//Välj element med Id "elementId"
var element = $("#elementId");

//Välj alla <span></span> objekt
var allaSpan = $("span");

//Vänta på att knappen med Id "clickMe" klickas, och gör något när detta inträffar
$("#clickMe").on("click", function(args){
    alert("Doing something!");
});

//Ajax-förfrågning
$.ajax({
  type: "POST",
  url: "example.php",
  data: "name=Robin&location=Vaasa"
}).done( function(msg) {
  alert( "Data Saved: " + msg );
}).fail( function( xmlHttpRequest, statusText, errorThrown ) {
  alert(
    "Your form submission failed.\n\n"
      + "XML Http Request: " + JSON.stringify( xmlHttpRequest )
      + ",\nStatus Text: " + statusText
      + ",\nError Thrown: " + errorThrown );
});
```

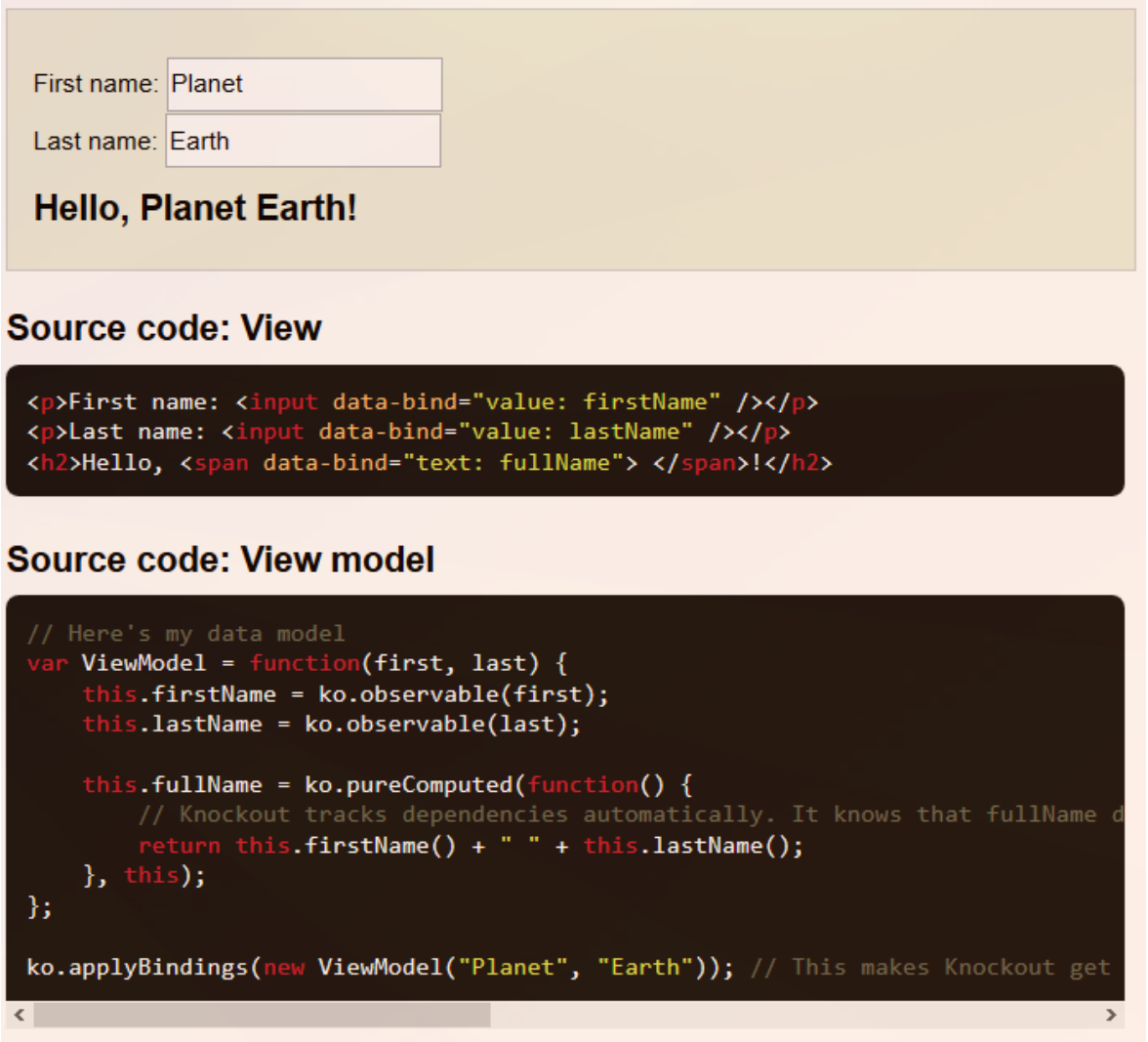
Kodexempel 3. Exempel på jQuery-syntax.

3.7.2 Knockout

Knockout är ett JavaScript-bibliotek, som följer utvecklingsprincipen Model-View-Viewmodel, och använder sig utav templates för att uppnå detta. Likt MVC vill man med detta uppnå en separering av databasens data, vyns komponenter och data som ska visas.

Funktionalitet som Knockout tillför, är bland annat följande:

- Håller koll på relationer mellan objekt, och uppdaterar relevanta delar av användargränssnittet när dessa ändrar.(se figur 5)
- Gör det möjligt att binda ett element i användargränssnittet till en tillhörande datamodell. Datamodellen kan dessutom innehålla ytterligare modeller, vilket möjliggör mera komplexa och inkapslade bindningar.



Figur 5. Skärmdump tagen av ett exempel på hur knockout-kod ser ut.[1]

Exemplet (se figur 5) demonstrerar det viktigaste knockout utför, nämligen databindningen. Om man ändrar värdet i något av fälten, och sedan klickar någon annanstans på sidan, uppdateras rubriktexten som börjar med ”Hello, ...”. Detta är möjligt tack vare att när sidan läses in, tilldelas sidan en viewmodel, ”ViewModel”, och viewmodellens objekt ”firstName” och ”lastName” blir bundna till varsitt paragrafelement. Uppdateras något av dessa paragrafelement, håller knockout reda på detta, och uppdaterar rubriktexten.

Man kan också skapa mera komplicerade bindingar, som till exempel när man önskar visa en tabell med flera rader, och möjligen också med flera sidor. För att åstadkomma detta kan man i viewmodellen använda sig av en array med objekt, och sedan i view använda en binding kallad foreach. Foreach bindningen använder sig utav ett template och skapar flera likadana element på sidan. (se figur 6) [20].

```
<table>
    <thead>
        <tr><th>First name</th><th>Last name</th></tr>
    </thead>
    <tbody data-bind="foreach: people">
        <tr>
            <td data-bind="text: firstName"></td>
            <td data-bind="text: lastName"></td>
        </tr>
    </tbody>
</table>
```

Figur 6. Skärmdump av ett exempel på knockouts foreach-bindning.[2]

3.7.3 RequireJS

RequireJS är ett JavaScript- bibliotek, som implementerar strukturering av kod i modul form, och inläsning av filer. [3]. Baserar sig på AMD API, som är en JavaScript-specifikation, och anger hur ett API, likt RequreJS, kodmässigt ska definierar moduler, och kräva andra moduler. [4][5].

Nyttan med att koda JavaScript i modulformat, är bland annat att man förkortar laddningen av en webbsida, tack vare att kod läses in efter behov. När man skapar en modul, anger man först ifall modulen kräver andra tidigare gjorda moduler. Ifall andra moduler krävs anges detta, och om modulen redan blivit inläst, finns logik i RequreJS som uppfattar detta och inte läser in modulen på nytt.

```
define([
    'module1',
    'module2'
],
    function (module1, module2) {

        return {
            sayHello: function () {
                alert("Hello!");
            }
        }

    });
```

Kodexempel 4. Bild på hur en modul i RequireJS ser ut.

Exemplet på bilden (se kodexempel 4) visar hur en modul byggs upp. ”define” är en global funktion, som finns tillgänglig efter att man inkluderat RequireJS biblioteket. Funktionen tar två argument, det första anger vilka moduler som fordras. Det andra argumentet är en

funktion som körs när denna modul blir inläst. Ofta definieras olika funktioner inuti modulen, som blir tillgängliga åt andra moduler som läser in denna modul. I detta fall funktionen ”sayHello”, som skriver ut ”Hello!”.

Eftersom man också anger att modulen kräver andra moduler, kan man vara säker på att funktioner man vill använda från den krävda modulen, finns tillgängliga. Detta i sin tur förbättrar felhantering.

Utöver detta blir struktureringen på koden märkbart mera hanterlig, och överskådlig. Moduler skapas oftast i separata filer, vilket betyder att uppdatering av endast en viss del av en applikation är mycket lättare.

3.7.4 Durandal

Durandal är ett JavaScript ramverk som helt körs i webbläsaren. Tack vare detta är Durandal tillgängligt på de flesta plattformar. Används för att utveckla Single Page Applikationer, och innehåller redan från början flera användbara JavaScript-bibliotek, såsom jQuery, Knockout och RequireJS.

Orsaker till att man vill använda sig av Durandal, kan vara någon eller några av följande:

- Överskådligt Model-View upplägg.
- Kod för både HTML och JavaScript hanteras i moduler.
- Finns inbyggd logik för att hantera navigering.
- Inbyggd funktionalitet för att skapa dialog, händelser, och informationsrutor.
- Man kan välja lämplig API teknik enligt egen önskan.

Durandal baserar sig på tvåvägsbundet användargränssnittsdata. Detta kan man ganska fort konstatera att det kommer från Knockout. Upplägget med HTML och JavaScript-kod i modulformat härstammar i sin tur från RequireJS. [6].

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="lib/bootstrap/css/bootstrap.css" />
    <link rel="stylesheet" href="lib/font-awesome/css/font-awesome.css" />
    <link rel="stylesheet" href="lib/durandal/css/durandal.css" />
    <link rel="stylesheet" href="css/starterkit.css" />
  </head>
  <body>
    <div id="applicationHost"></div>
    <script src="lib/require/require.js" data-main="app/main"></script>
  </body>
</html>
```

Figur 7. Durandals bas HTML-fil. [7].

Grundfunktionaliteten i en webbapplikation byggd med Durandal, börjar från en grund HTML fil, som i detta fall kallas ”index.html” (se figur 7). HTML filen är basen för själva webbapplikationen, och innehåller endast ett tomt element, som fylls med innehåll enligt behov. Det finns också kod som läser in några CSS-filer, och RequireJs biblioteket. När RequireJs biblioteket läses in, anges också i skript elementet sökvägen till Durandals grundkonfigurationsmodul, kallad ”main”, som behövs för att initiera resten av applikationen.

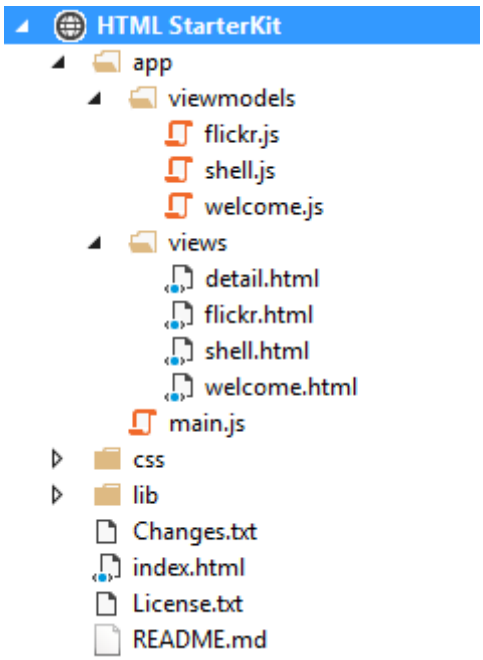
```
requirejs.config({
  paths: {
    'text': '../lib/require/text',
    'durandal':'../lib/durandal/js',
    'plugins' : '../lib/durandal/js/plugins',
    'transitions' : '../lib/durandal/js/transitions',
    'knockout': '../lib/knockout/knockout-2.3.0',
    'jquery': '../lib/jquery/jquery-1.9.1'
    }
});

define(function (require) {
   var system = require('durandal/system'),
       app = require('durandal/app');

   system.debug(true);

   app.title = 'Durandal Starter Kit';

   app.configurePlugins({
     router:true,
     dialog: true
   });

   app.start().then(function() {
     app.setRoot('shell');
   });
});
```

Kodexempel 5. Durandals main-modul. [7].

Main-modulen börjar först med att konfigurera RequireJs. Man ställer bland annat in vilka sökvägar alla nödvändiga moduler har, och dessa ges ett passande modulnamn. Detta görs för att senare lätt kunna kräva en modul med endast modulnamnet. Efter detta ställer man in

vilka olika Durandal-plugin man vill använda. Ett viktigt sådant är Routern, som hanterar navigeringen mellan applikationens sidor.

Efter att grundkonfigurationen gjorts, fortsätter applikationen med att ladda in en så kallad shell-modul, som innehåller kod som är gemensam över alla applikationens sidor. I shell modulen finns bland annat kod som hanterar navigationsmenyn, och eventuellt webbsidans sidfot.



Figur 8. Exempel på Durandals mappstruktur.

En sida i Durandal består av en HTML-fil (View), samt en JavaScript-fil (Viewmodel). Dessa bör ha exakt samma namn, men olika filtyp. För att hitta dessa, använder sig Durandal av en bestämd mappstruktur (se figur 8). När routern navigerar till en viss sida, läses båda dessa in. När detta händer, fylls grund HTML-filens tomma element (se figur 8) med det som finns i sidans HTML-fil. Modellen till sidan blir det som JavaScript-filen returnerar.

3.7.5 Breeze

Breeze är ett JavaScript bibliotek, som bland annat tillför dataförfrågningar, cachning av data, ändringsspårning, uppdatering av flera entiteter samtidigt, på klienten. [8].

När man traditionellt hämtar data från ett Web API och dess kontroller, blir man först tvungen att skapa metoder i denna kontroller för att returnera önskat objekt, eller en viss entitet. Man är också tvungen att skapa metoder för att radera eller editera en viss entitet. Detta gör man ofta för varje Modell man har. Breeze förenklar detta betydligt, och underlättar hantering av data.

I stället för att skapa alla dessa kontrollers, och deras metoder skilt, kan man till Web API projektet importera ett par installationspaket, som innehåller kod man behöver för att skapa en Breezekontroller. Breezekontrollern är väldigt lik en vanlig Web API kontroller, men den innehåller metoder för returnera metadata, spara ändringar, samt för varje modell, en metod som tar emot förfrågningar som gäller modell.

På klienten börjar allt från en EntityManager, som fungerar som en brygga till Breezekontrollern i Web API. När klienten först används, hämtas metadata från Web API, och information om alla tillgängliga entiteter sparas i EntityManagern. Denna information används senare för att skapa nya och editera existerande entiteter.

En entitet är ett objekt som innehåller alla egenskaper som databasmodellen har, men också information om eventuella relationer till andra entiteter, och egenskaper som anger om entiteten blivit ändrad. Entiteternas egenskaper är dessutom Knockout variabler, vilket betyder att dessa direkt kan bindas till kontroller i användargränssnittet.

Mot EntityManagern kan man göra förfrågningar, som kallas EntityQuery. Dessa ser relativt likadana ut som LINQ-förfrågningar (se kodexempel 6), vilket utvecklarna av Breeze troligen också haft som åtanke.

```
var query = breeze.EntityQuery()
    .from('Customers')
    .where('CompanyName', 'startsWith', 'C')
    .orderBy('CompanyName');

var manager = new breeze.EntityManager(serviceName);
manager.executeQuery(query)  // [1]
       .then(querySucceeded) // [2]
       .fail(queryFailed);   // [3]
```

Kodexempel 6. Exempel på en EntityQuery. [9].

Som figuren visar, körs EntityQueryn mot EntityManager. Vad Breeze egentligen utför, är att översätta förfrågningen till en HTTP GET- förfrågan, som Breezekontrollern i Web API

förstår. När Breezekontrollern får denna förfråga, måste den skrivas om, för att kunna köras mot databasen. För att göra detta används en komponent i .NET ramverket, kallad LINQ som innehåller tilläggsuttryck för programmerinsspråket, och gör att man kan skapa så kallade LINQ-uttryck. Responsen som skickas tillbaka till klienten formateras om till JSON format.

```
http://www.example.com/**api/Northwind/Customers?$filter=startswith(CompanyName,'C') eq true&amp;$orderby=CompanyName**
```

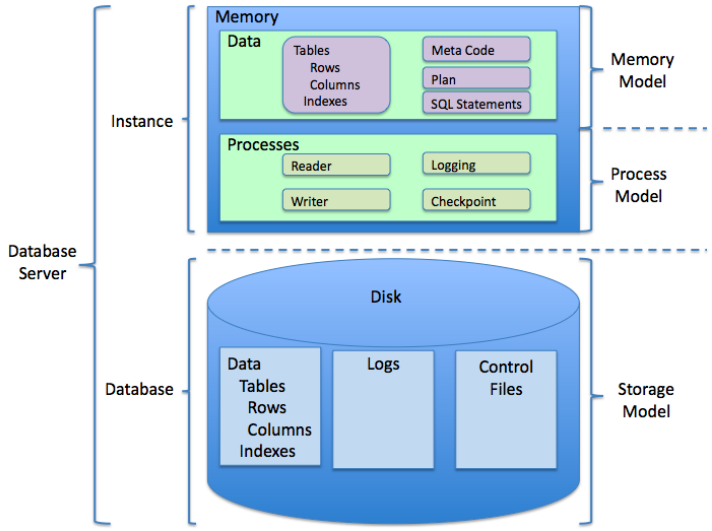
Figur 9. Exempel på HTTP GET förfråga som Breeze utför mot Web API. [9].

Samma tillvägagångssätt gäller för när man önskar spara en, eller flera entiteter till databasen. Skillnaden då är att entiteterna omvandlas till JSON på klienten, och skickas med i en HTTP POST – förfrågan. Adressen på denna är dock Breezekontrollerns ”SaveChanges” metod.

3.8 Microsoft SQL Server

Microsoft SQL Server är ett relationsbaserat databashanteringssystem, och har som uppgift att spara och hämta data som begärs av andra system. Dessa andra system kan beroende på fall finnas på samma fysiska server, eller på en annan dator över ett nätverk.

Liksom andra relationsbaserade databassystem, baserar sig SQL Server på en modell (se figur 10) som först uppfanns på 1969 av Edgar Frank Codd som arbetade för IBM.



Figur 10. Strukturen för ett relationsbaserat databassystem. [14].

Modellen ordnar data i en eller flera tabeller, som kan ses som relationer, och tabellerna innehåller kolumner och rader med unika nyckelvärden för varje rad. I vanliga fall används en tabell för att definiera en typ av entitet, till exempel ”Bil”. En rad i sammanhanget motsvarar en entitet av typen ”Bil”, till exempel ”Audi A4”.

En rad i en tabell kan också länkas ihop med en annan tabell, genom att använda en kolumn vars värde motsvarar det unika nyckelvärde i tabellen man vill länka ihop. Denna kolumn kallas då ”Foreign key”.

3.9 Entity Framework

Entity framework är en del av ADO.NET och innehåller tekniker som används för att underlätta utveckling av applikationer som kräver manipulation av data. Dessa tekniker kallas ORM, vilket står för Object-relation mapping. [19].

För att kunna hantera data som finns i en databas, har man tidigare varit tvungen, att i applikationskod skapa objekt som motsvarar databasens entiteter, och tillhörande funktioner som exekverar kommandon mot databassystemet i fråga. Varje enskild funktion för att läsa, skapa, eller modifiera en entitet i databasen har krävt ett skilt kommando (se figur 11).

```
String sql = "SELECT ... FROM persons WHERE id = 10";
DbCommand cmd = new DbCommand(connection, sql);
Result res = cmd.Execute();
String name = res[0]["FIRST_NAME"];
```

Figur 11. Exempel på icke Entity Framework funktion.

Det som Entity Framework erbjuder för att underlätta detta, är att det skapar en sorts virtuell databas, som kan användas från applikationens kod. Denna virtuella databas genererar, och returnerar så kallade domänobjekt med tillhörande attribut och funktioner, som liknar de som finns i grunddatabasen. Till exempel ett ”Bil”-objekt från databastabellen ”Bilar”.

Funktioner för att hämta alla objekt, eller göra förfrågningar som bara returnerar objekt som uppfyller vissa krav, finns färdigt implementerade och behöver inte skapas (se figur 12).

```
Person p = repository.GetPerson(10);
String name = p.getFirstName();
```

Figur 12. Motsvarande kommando till figur 11, men med Entity Framework.

Ändringar som görs på dessa objekt, eller ifall nya skapas, sparas i grunddatabasen. Uppdateringen av databasen, som kan använda sig av olika system, sköter Entity Framework

om med passande kommandon och syntax så länge man definierat vilken typ av databas man använder.

3.10 Microsoft Azure

Microsoft Azure är en molnbaserad samling tjänster som bland annat består av följande:

- Virtuella maskiner
- Databassystem
- Lagring
- Webb-, och mobilapplikationsvärd

För att kunna erbjuda en tjänst som använder sig utav ett databassystem, krävs i de flesta fall en värd, som ofta nämns ”Server” för detta. Tidigare var standarden oftast den, att företaget som utvecklat en webbapplikation hade en egen server som kunde användas som värd för databasen, och kanske också applikationen vid behov. Detta medförde ytterligare kostnader, såsom införskaffning, elförbrukning och underhåll. Dessutom vill man också ha någon sorts backup på systemet.

Tack vare att datorer blivit kraftigare, lagring betydligt billigare och internet-förbindelser snabbats upp och blivit pålitligare, finns molntjänster till ett väldigt konkurrenskraftigt pris.

Microsoft Azure hanteras från en webbportal, där man får en överblick över vilka tjänster som man använder, samt kan lägga till tjänster efter behov. Vill man till exempel erbjuda en webbapplikation, skapar man en sådan med tillhörande databas på bara några minuter. Efter detta laddar man upp sin kod, och applikationen är i gång och kan användas på internet.

Applikationen finns nu på en server som Microsoft Azure underhåller och man får ett löfte om att servern är tillgänglig 99,9 % av tiden.

4 Verktyg

Detta kapitel kommer att ytligt presentera några verktyg och hjälpmedel som använts för att skapa applikationen och hantera applikationens databas.

4.1 Visual Studio 2013

Microsoft Visual Studio är en integrerad utvecklingsmiljö, som används för att skapa program för Windows, webbsidor, webbapplikationer och webbtjänster. Visual Studio innehåller också stöd för många vanliga utvecklingsplattformer, till exempel Microsoft Silverlight och Windows Store. [15].

Den mest använda funktionen i Visual Studio är kod-editorn. Kod-editorn innehåller en sorts intelligens, som kallas Intellisense, och underlättar vid kodning. Intellisensen ger förslag på till exempel vilka metoder ett visst objekt har, så att man inte behöver memorera den exakta syntaxen. Andra användbara funktioner är till exempel webbdesignern, klassdesignern och databas schema designern.

Visual Studio har integrerad support för de vanligaste programmeringsspråken, såsom C, C++, C# med flera, och de som inte finns går att installera.

4.2 Visual Studio Team Services

Visual Studio Team Services är ett verktyg som kan användas av projekt team, när man behöver samarbeta vid utveckling av program. För att använda VS Team Services, registrerar man sig på deras sida, och skapar en team-sida dit alla medlemmar läggs till. På team-sidan kan man sedan skapa projekt, och välja vilka medlemmar som hör till ett visst projekt. Efter detta kan man i Visual Studio välja att ansluta sig till ett eller flera projekt.

När man anslutit sig till ett projekt, får man tillgång till kod som finns i projektmappen, eller så kan man ladda upp sin egen kod. Kod som laddas upp blir då till en sorts backup som kan kommas åt över internet. Efter att man gör ändringar i filerna som man ursprungligen laddat upp, behålls en historik och man kan gå tillbaka till tidigare versioner vid behov.

Om ett team skulle använda denna funktionalitet, kunde man foga ihop ändringar som alla medlemmar gjort, och få en bra överblick över dessa ändringar. Alla nya versioner som laddas upp, kan även kommenteras.

5 Utförande

Detta kapitel kommer att presentera det praktiska utförandet av vad detta examensarbete handlar om. Planering, utveckling och en djupare beskrivning och förklaring av de olika skedena och funktionerna som applikationen har.

5.1 Planering

Vid projektets början fanns redan en projektbeskrivning som bestod av en PowerPoint presentation, med några sidor. Denna presentation hade skapats vid ett tidigare tillfälle, och var tillsammans med möjligheten att fråga av uppdragsgivare, den utgångsinformation som fanns.

Utifrån detta kunde ett grovt databasschema skapas. Hur detta gjordes, var genom att gå igenom presentationen, och markera viktiga ord i meningar. Utifrån dessa markeringar kunde man relativt snabbt bygga upp tabeller i Excel, som användes som utgångspunkt (se figur 13).

Waybill	**Customer**	**CustomerRegister**
Id	Id	Id
CreatedOn	Name	Name
WorkOrderId	Address	Address
SenderId(Customer)	PostalCode	PostalCode
RecieverId(Customer)	City	City
Freight (Id)	PhoneNumber	PhoneNumber
	CustomerRegisterId	EmailAddress
	EmailAddress	
Freight	**Invoice**	**WorkOrder**
Id	Id	Id
ParcelAmount	CreatedOn	SenderId(Customer)
Weight	Status	RecieverId(Customer)
Volume	WorkOrderId(Workorder)	PayerId(Customer)
Content	Barcode	CreatedOn
DeliveryClause	InvoiceNo	Freight (Id)
Package		

Figur 13. Databasschema.

Tanken när schemat gjordes, var att utgå från en entitet som skulle kallas **Order** (WorkOrder), dvs. ordern som speditören tar emot när denna får ett samtal. En order behövde därför innehålla information om tre olika kunder, i form av avsändare (Sender), mottagare (Reciever), och betalare (Payer). För att få en bättre struktur, och inte fylla order objektet med kundinformation, skapades en till typ av objekt vid namn kund (Customer). Efter att detta hade gjorts, kunde avsändare, mottagare, och betalare egenskaperna i ordern länkas ihop, genom att ange ett specifikt ID-nummer till respektive kund.

För att inte behöva fylla i all information för kunderna när en ny order matas in, skapades också ett kundregister (CustomerRegister). Kundregistret innehåller information om en kund, som vid inmatning av en ny order kopieras till en ny kund. Orsaken till att informationen kopieras, och inte länkas ihop, är att en kund i registret kanske i något skede

byter adress, och man vill inte att leveransadressen på tidigare körningar ska ändra. Detta kunde till exempel göra fakturor ogiltiga.

Nästa objekttyp som behövde finnas i applikationen var en frakt (Freight). En frakt innehåller information om det som speditören fraktar, till exempel vikt, kolliantal, volym osv. Också ett fraktregister skapades senare, men i planeringsskedet hade detta glömts bort.

Faktura (Invoice) och Fraktsedel (Waybill) objekten, som fanns med på det ursprungliga schemat konstaterades ganska fort vara obehövliga. Orsaken till detta var att fraktsedeln var något som genererades utifrån orderinformationen, och fraktsedelobjektet tillförde inget. Det samma gällde för fakturan. Istället tillsattes en egenskap till ordern, som anger om ordern blivit fakturerad eller inte.

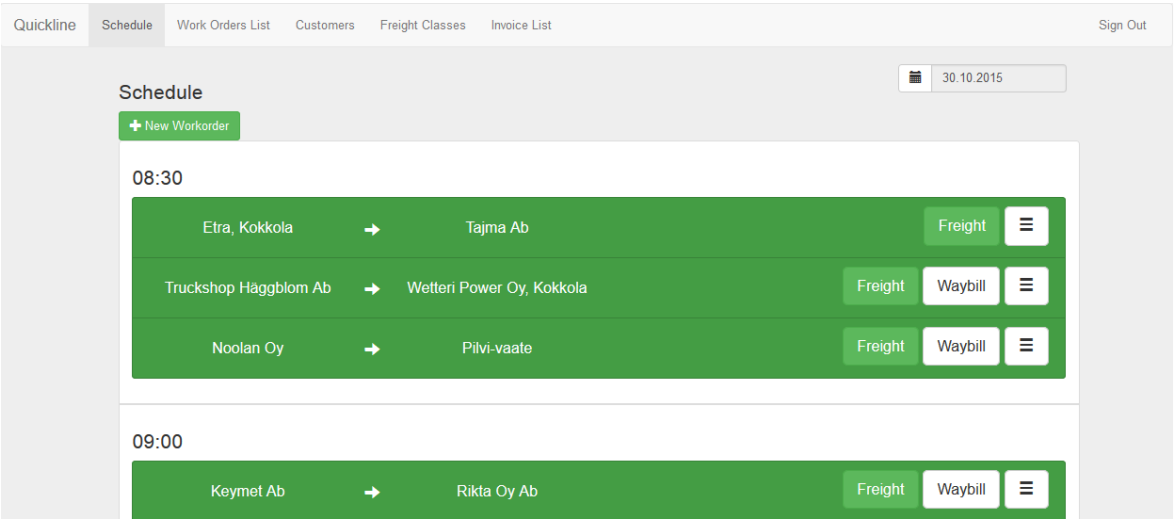
5.2 Utveckling

Utvecklingen av applikationen började med att studera exempel som hade påminnande funktionalitet. Ett av exemplen hittades i en bok [16], och blev utgångspunkten. Exemplet i fråga skildrade uppbyggnaden av en webbutik för sportartiklar. De flesta stegen följdes, tills en ganska klar uppfattning över funktionerna för de olika delarna i applikationen hade uppnåtts.

Efter detta blev nästa steg att ändra om benämningarna på de olika modellerna, vyerna och kontrollerna. Strukturen av applikationen bibehölls till största delen som den i exemplet. I detta tidiga skede i utvecklingen fanns ingen vidare planering över hur själva användargränssnittet skulle se ut, utan detta utarbetades vid sidan om att själva logiken för applikationen byggdes upp.

5.2.1 Körschema

Den första sidan, eller egentligen vyn eftersom inga sidor finns i en SPA, som skapades var körschemat (se figur 14).



Figur 14. Körschema.

Körschemat är uppbyggt så att alla körningar är grupperade enligt vilket klockslag för upphämtning de har. Klockslaget är ungefärligt, och användaren kan inom gruppen sortera körningarna enligt den mest lönsamma rutten, genom att dra och släppa dessa.

På sidan finns ett antal knappar som kan användas för att snabbt mata in tilläggsinformation bland annat;

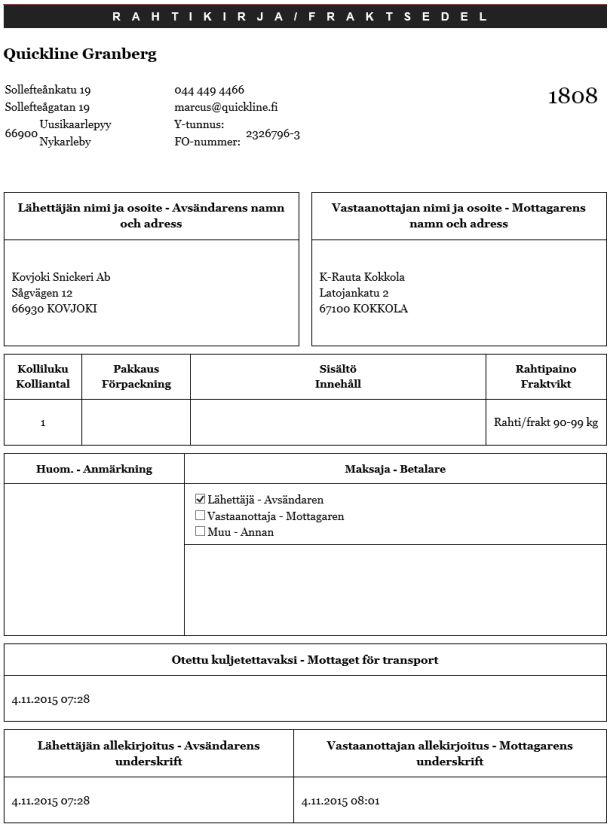
- New Workorder – Mata in en ny körning.
- Freight – Mata in, eller ändra fraktinformation.
- Waybill – Printa ut en fraktsedel.

Knappen längst till höger under en körning, innehåller också genvägar till att mata in en kunds eget fraktsedelnummer, samt markera en körning som antingen upphämtad eller levererad. En körnings status representeras av vilken bakgrundsfärg den har. Vit betyder att den endast blivit inmatad, orange betyder att körningen blivit upphämtad, och grön att körningen blivit levererad. Också färgen på Freight-knappen har en liknande betydelse, där orange betyder att ingen fraktinformation finns, och grön att fraktinformation blivit inmatad.

5.2.3 Fraktsedel

Fraktsedelsidan är en helt separat sida inom webbapplikationen. Med detta menas att MVC kontrollern som returnerar grundsidan för SPA:n, också returnerar en annan sida i form av fraktsedelsidan. Detta görs på grund av att sidan inte ska innehålla någon design eller logik

från huvudsidan, utan all egen kod och design finns, för att skapa en sida som liknar en fraktsedel (se figur 15).



RAHTIKIRJA / FRAKTSEDEL

Quickline Granberg

Sollefteånkatu 19
Sollefteågatan 19
66900 Uusikaarlepyy
Nykarleby

044 449 4466
marcus@quickline.fi
Y-tunnus:
FO-nummer: 2326796-3

1808

Lähettäjän nimi ja osoite - Avsändarens namn och adress	Vastaanottajan nimi ja osoite - Mottagarens namn och adress
Kovjoki Snickeri Ab Sågvägen 12 66930 KOVJOKI	K-Rauta Kokkola Latojankatu 2 67100 KOKKOLA

Kolliluku Kolliantal	Pakkaus Förpackning	Sisältö Innehåll	Rahtipaino Fraktvikt
1			Rahti/frakt 90-99 kg

Huom. - Anmärkning	Maksaja - Betalare
	☑ Lähettäjä - Avsändaren ☐ Vastaanottaja - Mottagaren ☐ Muu - Annan

Otettu kuljetettavaksi - Mottaget för transport
4.11.2015 07:28

Lähettäjän allekirjoitus - Avsändarens underskrift	Vastaanottajan allekirjoitus - Mottagarens underskrift
4.11.2015 07:28	4.11.2015 08:01

Figur 15. Fraktsedelsidan.

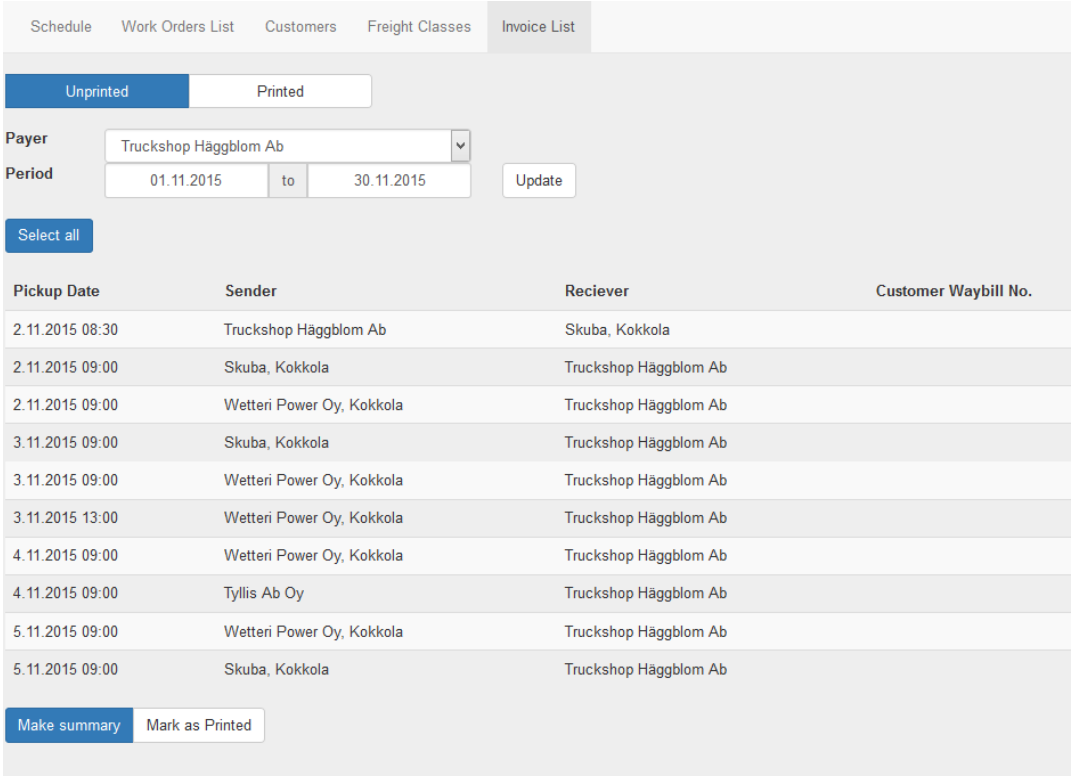
Ett av kraven vid början av projektet var att kunna skriva ut en fraktsedel. Till först undersöktes ett alternativ som gick ut på att Web API skulle generera och returnera en PDF fil tillbaka till webbläsaren. Några tillägg för detta testades, men något vettigt resultat blev aldrig av. Man skulle hur som helst bli tvungen att skapa en modell för fraktsedeln, och ungefär samma arbetstid skulle gå åt till att göra en simpel HTML-sida, som kunde användas för ändamålet.

Logiken som finns på sidan är inte så avancerad. När användaren klickar på en orders fraktsedelknapp, skickas information om ordern till fraktsedelsidan. Denna information läses in, och med hjälp av knockout och dess databindingar fylls relevanta fält i. Till exempel namnet på avsändaren och mottagaren, fraktinformation osv.

Högst uppe på sidan finns en knapp, som med hjälp av JavaScript öppnar en printdialog i webbläsaren. Från dialogen blir användaren tvungen att välja att inte skriva ut några marginaler i dokumentet, varefter utskriften skrivs ut som sidan ser ut i exemplet ovan.

5.2.4 Fakturasammanfattning

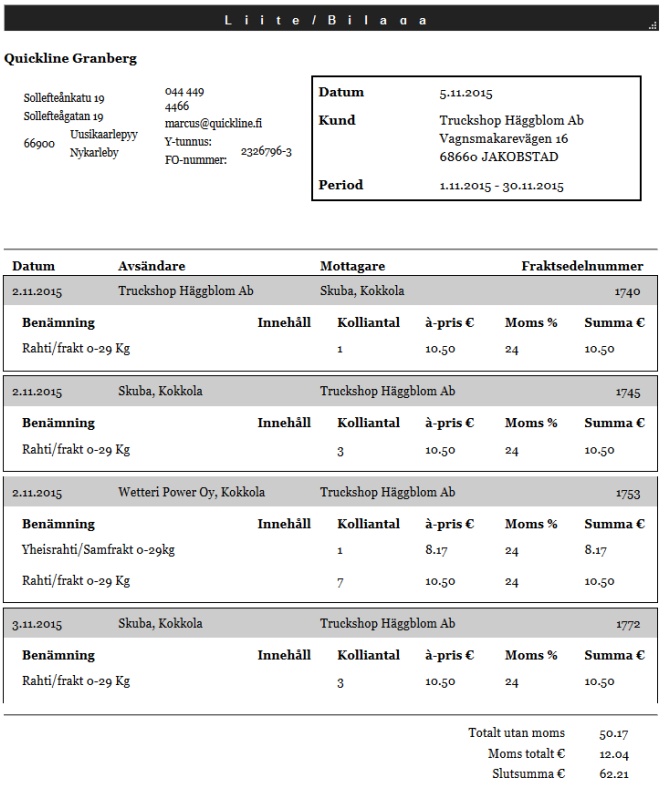
Fakturasammanfattningssidan (se figur 16) innehåller egentligen två identiska undersidor, som hanterar ordrar med status oprintad, respektive printad. Sidan används för att skapa en bilaga över alla ordrar som gjorts för en viss betalare, för en viss tid, som används vid fakturering.



Figur 16. Fakturasammanfattningssidan.

Användningen av sidan börjar med att användaren väljer en betalare ur en dropdown-lista, som automatiskt fylls i, enligt vilka betalare som har obetalda ordrar. När betalare valts väljs en period, med ett start- och slutdatum. Efter detta hämtas alla relevanta ordrar som motsvarar dessa kriterier från databasen, och en tabell fylls i. Från tabellen kan användaren välja, genom att markera enskilda eller alla ordrar, vilka som ska finnas med i bilagan.

Efter att dessa valts, och man klickat på knappen ”Make summary”, öppnas en ny sida som kan printas ut och användas som bilaga (se figur 17). Denna bilaga är likt fraktsedeln, en helt skild sida utanför SPA-grundsidan, och returneras av MVC-kontrollern. På sidan finns en del logik, som räknar ihop totala priset, moms, och fyller i kundinformation.



Liite / Bilaga

Quickline Granberg

Sollefteånkatu 19
Sollefteågatan 19
66900 Uusikaarlepyy
Nykarleby

044 449
4466
marcus@quickline.fi
Y-tunnus:
FO-nummer: 2326796-3

Datum	5.11.2015
Kund	Truckshop Häggblom Ab Vagnsmakarevägen 16 68660 JAKOBSTAD
Period	1.11.2015 - 30.11.2015

Datum	Avsändare	Mottagare	Fraktsedelnummer
2.11.2015	Truckshop Häggblom Ab	Skuba, Kokkola	1740

Benämning	Innehåll	Kolliantal	à-pris €	Moms %	Summa €
Rahti/frakt 0-29 Kg		1	10.50	24	10.50

2.11.2015	Skuba, Kokkola	Truckshop Häggblom Ab	1745

Benämning	Innehåll	Kolliantal	à-pris €	Moms %	Summa €
Rahti/frakt 0-29 Kg		3	10.50	24	10.50

2.11.2015	Wetteri Power Oy, Kokkola	Truckshop Häggblom Ab	1753

Benämning	Innehåll	Kolliantal	à-pris €	Moms %	Summa €
Yheisrahti/Samfrakt 0-29kg		1	8.17	24	8.17
Rahti/frakt 0-29 Kg		7	10.50	24	10.50

3.11.2015	Skuba, Kokkola	Truckshop Häggblom Ab	1772

Benämning	Innehåll	Kolliantal	à-pris €	Moms %	Summa €
Rahti/frakt 0-29 Kg		3	10.50	24	10.50

Totalt utan moms	50.17
Moms totalt €	12.04
Slutsumma €	62.21

Figur 17. Fakturerings bilaga.

Efter att användaren printat ut bilagan, används knappen ”Mark as Printed”, (se figur 16) för att ändra status för de markerade ordrarna.

5.2.5 Testning

När applikationen fanns i en första version, och hade grundfunktionalitet, bestämdes med speditören att applikationen skulle tas i bruk parallellt med speditörens papperssystem. Detta gav strax mycket feedback, som hjälpte till för vidareutvecklingen av applikationen. Det visade sig dock att speditören helt hade övergått till det nya systemet, efter den första dagen av testet. Detta kan kanske tolkas som om speditören genast fick ett bra intryck av applikationen.

5.3 Applikationens uppbyggnad

Detta kapitel beskriver applikationens kodstruktur, samt hur de olika delarna hör ihop och kompletterar varandra. Med detta menas de olika delarna som tillsammans utgör den fullständiga applikationen, såsom databasen, Web-API:t och klienten.

5.3.1 Databasen

Databasen är en SQL-databas, som fysiskt finns i molnet, i tjänsten Microsoft Azure. Denna databas är funktionsmässigt väldigt lik en vanlig SQL-databas, som man kan installera och använda lokalt på de flesta datorer.

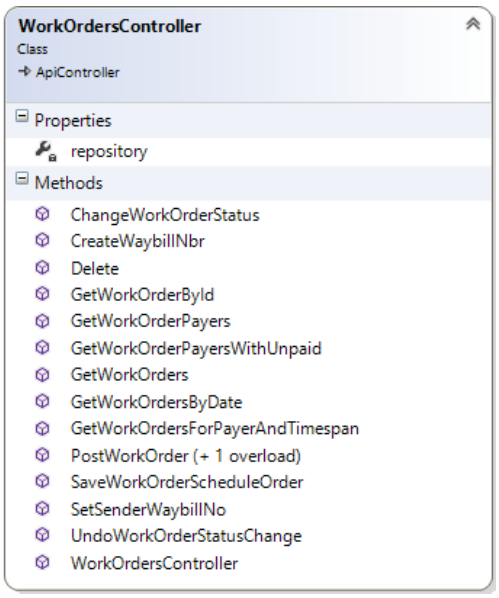
För att komma åt databasen, krävs att i Web-API:t ställer in en adress för databasen, med rättigheter. Detta görs i konfigurationen för Web-API:t, i form av att man ändrar en textsträng, som kallas för databasens Connectionstring (se kodexempel 7). När detta har gjorts har Web-API:t kontakt till databasen, och databasens fysiska plats spelar ingen större roll. Om man önskar kan databasen finnas på den egna maskinen, eller på en server utomlands.

```
<connectionStrings>
  <add name="QuicklineIdentityDbContext"
       providerName="System.Data.SqlClient"
       connectionString="
       Server=tcp:example.database.windows.net,1433;
       Database=example_db;User ID=example@exampleServer;
       Password=******;Trusted_Connection=False;Encrypt=True;
       Connection Timeout=30;" />
</connectionStrings>
```

Kodexempel 7. Exempel på en connectionstring.

5.3.2 Web API

Web API:t är komponenten i applikationen som får alla delar att spela ihop. När klienten använder sig utav Web API, gör den detta genom att anropa olika controllers. Web API:t är uppbyggt så att alla så kallade entiteter, eller värdetyper har varsin controller (se figur 18).



Figur 18. Exempel på en controller och dess metoder.

Varje controller har sina metoder, som klienten använder, för att göra ändringar på den typ av entitet som controllern hanterar. Det är oftast frågan om att hämta, skapa, modifiera eller radera en viss entitet. Alla controllers har en koppling till en så kallad repository, som är en komponent som innehåller färdig funktionalitet för att läsa och skriva till databasen. Denna färdiga funktionalitet är det som Entity Framework erbjuder (se kapitel 3.9)

Web API:t finns fysiskt, likt databasen i molnet, på Microsoft Azure.

5.3.3 Webbapplikationen

Med webbapplikationen menas i detta sammanhang, ett program som körs på en server, och returnerar kod i form av HTML och JavaScript till webbläsaren. Denna kod som returneras representerar den så kallade klientapplikationen som körs i webbläsaren.

Webbapplikationen är till konstruktionen väldigt enkel, utan någon avancerad logik, men behövs för att distribuera själva klientapplikationen till användaren. Detta sker när användaren med sin webbläsare, navigerar till adressen för applikationen. Eftersom denna webbapplikation är av typen MVC (se kapitel 3.1), betyder det att man hade kunnat bygga upp hela applikationen endast genom att använda MVC, och inte bygga någon klientapplikation. Skillnaden i det fallet hade varit att, all mera betydande logik hade utförts på servern istället för i webbläsaren.

Men eftersom SPA lät intressant och är på uppgång, föll valet på att skapa en sådan.

5.3.4 Klientapplikationen

Klientapplikationen är den HTML, och JavaScript kod som körs i webbläsaren. Enligt Durandals uppbyggnad, och principer (se kapitel 3.4.4) har varje så kallad ”Sida” i applikationen, dvs. Schedule, Workorders, Customers osv. varsin HTML- respektive JavaScript-fil. Dessa läses in beroende på vart användaren navigerar, till exempel skriver man in ”http://applikation.test/#Schedule”, hanterar Durandals inbyggda router detta genom att läsa ut den sista biten ur adressen efter hästhagen, dvs. ”Schedule”, och returnerar Schedule-sidan.

Utöver dessa filer som representerar klientens ”Sidor”, finns också ett par hjälpmoduler som innehåller funktioner för att anropa Web API:t, och hantera autentikering. En sådan modul, kallas apioperator, och innehåller metoder för att hämta, uppdatera och ta bort olika entiteter. Kodexempel 8, visar en del av dessa metoder för att ge en liten överblick. Till exempel när

man på någon sida vill visa alla WorkOrders, anropar man apioperator- modulens getWorkOrders funktion, som i sin tur bygger upp en query som ska köras mot Web Api, genom att använda Breeze.

```
define(['jquery', 'knockout', 'services/security', 'breeze'],
    function ($, ko, security, breeze) {
        var apioperator = {
            getWorkOrders: function() {
                var query = breeze.EntityQuery.from("WorkOrders");
                return manager.executeQuery(query);
            },
            getWorkOrder: function(id){
                var query = breeze.EntityQuery.from("WorkOrders").where("Id", "==", parseInt(id));
                return manager.executeQuery(query);
            },
            saveWorkOrder: function () { /*.....*/ },
            removeWorkOrder: function () {/*.....*/}
        }
        return apioperator;
    });
```

Kodexempel 8. Exempel på apioperator modulen.

6 Resultat och diskussion

Detta kapitel kommer att först framföra resultatet av detta examensarbete. Efter detta tas problem som uppstod under arbetets gång upp och hur dessa löstes. I kapitlet tas också upp hur applikationen kunde utvecklas. Slutligen förs en diskussion över vad skribenten lärt sig under arbetets gång.

6.1 Resultat

Resultatet blev en applikation som har tagits i bruk och dagligen används av speditörfirman. Speditören har till största del varit nöjd med applikationen, och har ytterligare önskemål om vidareutveckling, såsom möjlighet för mottagaren att skriva under direkt i applikationen.

6.2 Problem

Problem som uppstod under utvecklingen av applikationen, var relaterade till webbteknikerna som användes. Inledningsvis så användes Durandal inte i början av utvecklingen. Detta betydde att all html-kod som bygger upp sidan fanns i en enda html-fil. Efter att man i applikationen utförde vissa navigeringar, eller handlingar, doldes och visades sektioner enligt behov. Detta var i sig inget problem, men att bygga upp denna logik, och sedan göra uppdateringar blev ganska fort väldigt svårt och tungt. Detta gjordes senare

betydligt lättare med Durandal, som färdigt innehåller denna logik. Men det hade underlättat att ha känt till om Durandal i ett tidigare skede.

Andra småproblem dök upp under användningen av applikationen. Dessa var inte så många, och heller inte så betydande. Oftast var det endast frågan om att någon inmatning som användaren gjorde, innehöll något fel och inte kunde sparas. Som exempel kan nämnas när en ny kund, som inte hade e-postadress skulle skapas, och man var tvungen att mata in en e-postadress. Relativt lätt fixade småsaker.

6.3 Vidareutveckling

Några månader efter att applikationen hade tagits i bruk, gjordes en version två som byggde på Durandal. Genom att detta gjordes blev applikationen mera modulbaserad och anpassningsbar. Tanken med detta var att uppdragsgivaren skulle kunna marknadsföra applikationen åt andra liknande speditörsfirmor, och kunna anpassa sig till deras krav på ett bättre sätt.

Detta har inte ännu fullt ut blivit verklighet, men ett par liknande webbapplikationer har skapats för andra scenarion, och dessa kunde kanske ses som en sorts vidareutveckling på frakt-applikationen.

6.4 Diskussion

Slutligen måste jag nämna att examensarbetet, samt detta projekt har lärt mig väldigt mycket, inte bara när det gäller teknikerna som använts, utan också allt annat som hör till ett projekt. Små saker som hur användaren först skred till användning av en sida, kan variera stort från hur man själv föreställt sig att sidan skulle användas. Man hade liksom tagit på sig skygglappar, och koncentrerat sig på att så här kommer användaren att göra, och inte alls tänkt att så kan man också göra.

När projektet kändes som att det nästan stannat upp, och man hade möte med kunden och fick en sådan upplevelse, gjorde att man fick många nya idéer och kunde gå vidare.

7 Källförteckning

[1] Knockout [Online]
http://knockoutjs.com/examples/helloWorld.html [hämtat: 19.10.2015]
[2] Knockout [Online]
http://knockoutjs.com/documentation/foreach-binding.html [hämtat: 20.10.2015]
[3] RequireJS [Online]
http://requirejs.org/ [hämtat: 20.10.2015]
[4] RequireJS [Online]
http://requirejs.org/docs/whyamd.html [hämtat: 20.10.2015]
[5] Asynchronous module definition [Online]
https://en.wikipedia.org/wiki/Asynchronous_module_definition [hämtat: 20.10.2015]
[6] Durandal [Online]
http://durandaljs.com [hämtat: 21.10.2015]
[7] Durandal [Online]
http://durandaljs.com/get-started.html [hämtat: 21.10.2015]
[8] Breeze [Online]
http://www.getbreezenow.com/breezejs [hämtat: 22.10.2015]
[9] Breeze [Online]
http://breeze.github.io/doc-js/ [hämtat: 23.10.2015]
[10] HTML
https://en.wikipedia.org/wiki/HTML [hämtat: 23.10.2015]
[11] Cascading Style Sheets
https://en.wikipedia.org/wiki/Cascading_Style_Sheets [hämtat: 26.10.2015]
[12] Mozilla Developer Network
https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript
[hämtat: 26.10.2015]
[13] JSON
http://json.org/ [hämtat: 28.10.2015]
[14] Relational database management system
https://en.wikipedia.org/wiki/Relational_database_management_system#/media/File:RDB
MS_structure.png [hämtat: 16.11.2015]
[15] Microsoft Visual Studio
https://en.wikipedia.org/wiki/Microsoft_Visual_Studio [hämtat: 25.11.2015]

[16] Freeman, A., 2014. *Expert ASP.NET Web API 2 for MVC Developers*. New York: Apress. S. 58, Figur 4-2.

[17] Galloway, J., Wilson, B., Scott Allen, K. & Matson, D., 2014. *Professional ASP.NET MVC 5*. Indianapolis: Wrox. s. 2-3.

[18] Model–view–controller https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller [hämtat 13.03.2016]

[19] Mueller, J. P. 2013 *Microsoft ADO.NET Entity Framework Step by Step.* California: O´Reilly Media, Inc. S. 3-7.

[20] Munro, J. 2015. *Knockout.js Building Dynamic Client-Side Web Applications.* California: O´Reilly Media, Inc. S. 1-4.